Relational Database Models

Basic Concepts
Relational Theory

Basic concepts of the relational model 1

- relation, attribute, tuple
  cf file, field type, record occurrence

relations have a **degree** (= # of attributes)
and **cardinality** (= # of tuples)

intensional view of relation = time-independent aspect
extensional view = current state of relation contents

- keys: primary, candidate, alternate

Basic concepts of the relational model 2

Relation = special kind of file?

1. every 'file' contains only one record type
2. each record occurrence in a 'file' has same number of fields cf "OCCURS DEPENDING ON” in COBOL
3. each record occurrence has a unique identifier
4. within a 'file', record occurrences have an unspecified ordering, or are ordered according to values assoc with occurrences (needn't be by primary key)

Basic concepts of the relational model 3

Constraints

Key constraints

i.e. constraints implied by the existence of candidate keys (as specified in DB intension)

- uniqueness of tuples with given key
- attributes in primary keys non-null

Basic concepts of the relational model 4

Constraints ...

Referential constraints

Intension (indirectly) gives a specification of foreign keys in a relation (as in the supplier-parts relation, with tuples of the form (S#, P#, QTY))

The use of keys for supplier and parts in this way independently constrains the S# and P# attributes to values that are either null or designate uniquely identified entities

Basic concepts of the relational model 5

Constraints …

Integrity constraints

Certain constraints are imposed by the semantics of the data. E.g. person's height is positive, date-of-birth won't normally be a future date etc

Real-world constraints can be too rich to express:

- hard to capture type of real-world observables
- have data dependent constraints, motivating triggers

Summary: Basic concepts of relational model

Relation: relation, attribute, tuple

Relation as analogue of file: cf. file, field type, record

Relational scheme for a database: cf. file system

- Degree and cardinality of a relation

- Intensional & extensional views of a relational scheme

Keys: primary, candidate, foreign

Constraints: key, referential, integrity

Query Languages for Relational Databases 1

Issue: how do we model data extraction formally?

E.F. ("Ted") Codd is the pioneer of relational DBs
Early papers: 1969, 70, 73, 75

Two classes of query language: algebra / logic

1. Algebraic languages
   a query = **evaluating an algebraic expression**

2. Predicate Calculus languages
   a query = **finding values satisfying predicate**

Query Languages for Relational Databases 2

Issue: how do we model data extraction formally?

2. Predicate Calculus languages
   a query = **finding values satisfying predicate**

Two kinds of predicate calculus language

Terms (primitive objects) tuples xor domain values:

- tuples $\Rightarrow$ tuple relational calculus
- domain values $\Rightarrow$ domain relational calculus

Query Languages for Relational Databases 3

Examples of Query Languages

algebraic: ISBL - Information System Base Language

tuple relational calculus: QUEL, SQL

domain relational calculus: QBE - Query by Example

Issue: how are these languages to be compared?

Query Languages for Relational Databases 4

Issue: how are query languages to be compared?

Answer (Codd)

Can formulate a notion of **completeness**, and show that the core queries in these languages have equivalent expressive power

- mathematical notion, based on **relational algebra**
- in practice, is a *basic* measure of expressive power: practical query languages are ‘more than complete’

Relational Algebra 1

Relational Algebra

algebra = underlying **set** with **operations** on it

elements of the underlying set are referred to as
"elements of the algebra"

relational algebra = set of relations + ops on relations

cf set of polynomials with addition and multiplication

Relational Algebra 2

… relational algebra = set of relations + ops on relations

Definition: a (mathematical) relation
is a subset of $D_1 \times D_2 \times .... \times D_r$
where $D_1, D_2, ..., D_r$ are **domains**

Typical element of a relation is $(d_1, d_2, ..., d_r)$
where $d_i \in D_i$ for $1 \Omega i \Omega r$

$D_1 \times D_2 \times .... \times D_r$ is the **type** of the relation

r is the **arity** of the relation

Relational Algebra 3

Mathematical relation is an abstraction

- types are restricted to mathematical types
  e.g. height, weight and currency all numerical data
- components of a mathematical relation are indexed
  don't use named attributes in the mathematical treatment - in effect, named attributes just make it more convenient to specify relational expressions

.... ‘abstract’ expressive power unchanged

Relational Algebra 4

Basic algebraic operations on relations

1. Union

$R \cup S$ defined when R and S have same type
$R \cup S$ = union of the sets of tuples in R and S

2. Set Difference

R – S defined when R and S have same type
R – S is the set of tuples in R but not in S

Relational Algebra 5

Basic algebraic operations on relations ...

3. Cartesian Product

R of type $D_1 \times D_2 \times .... \times D_r$
S of type $E_1 \times E_2 \times .... \times E_s$

R × S is of type $D_1 \times D_2 \times .... \times D_r \times E_1 \times E_2 \times .... \times E_s$

R × S is the set of tuples of the form
$(d_1, d_2, ..., d_r, e_1, e_2, ..., e_s)$
where $(d_1, d_2, ..., d_r) \in R$, $(e_1, e_2, ..., e_s) \in S$

Relational Algebra 6

Basic algebraic operations on relations …

4. Projection

$\Pi_{i(1), i(2), ..., i(t)}$ (R) is defined whenever R has arity r and i(j)'s are distinct indices with 1 Ω i(j) Ω r for 1 Ω j Ω t

For a tuple, projection is defined by

$\Pi_{i(1), i(2), ..., i(t)} (d_1, d_2, ..., d_r) = (d_{i(1)}, d_{i(2)}, ..., d_{i(t)})$

$\Pi_{i(1), i(2), ..., i(t)}$(R) = set of *distinct* projections of tuples in R

Relational Algebra 7

Basic algebraic operations on relations …

5. Selection

Let F be a logical propositional expression made up of elementary algebraic conditions.

σ_F(R) is the set of tuples t in R whose components satisfy the condition F(t).

In the absence of attribute names, refer to components of tuples by index in F

e.g. $\sigma_{1=\text{"London"} \vee 1=\text{"Paris"}}$ (R) refers to set of tuples whose first component is either London or Paris

Relational Algebra 8

Simple examples of basic operations

R :

x	y	z
a	b	c
a	y	c

S :

x	y	t
a	b	c

R ∪ S : *union*

x	y	z
a	b	c
a	y	c
x	y	t

R – S: *difference/minus*

x	y	z
a	y	c

Relational Algebra 9

Simple examples of basic operations ...

R :

x	y	z
a	b	c
a	y	c

S :

x	y	t
a	b	c

R × S : *cartesian product*

x	y	z	x	y	t
x	y	z	a	b	c
a	b	c	x	y	t
a	b	c	a	b	c
a	y	c	x	y	t
a	y	c	a	b	c

Relational Algebra 10

Simple examples of basic operations …

$\prod_{2,3}(R)$:

y	z
b	c
y	c

R :

x	y	z
a	b	c
a	y	c

$\prod_{3}(R)$: z c *projection*

… note that duplicates are deleted

$\sigma_{1=\text{"a"}}(R)$: *selection*

a	b	c
a	y	c

Relational Algebra 11

Summary of basic operations …

1. Union	$R \cup S$
2. Set Difference	$R - S$
3. Cartesian Product	$R \times S$
4. Projection	$\prod_{i(1), i(2), ..., i(t)}(R)$
5. Selection	$\sigma_F(R)$

Codd’s definition of **completeness:**

a query language is **complete** if it can simulate all 5 basic operations on relations

Relational Algebra 12

Use of attribute names

In practical use of query languages, commonly use attribute names to define operations, e.g.

- projection onto specific attribute names
- identification of components in selection
- making distinctions between domains
- forming natural joins

Claim:

none of these devices specifies operations that can't be derived from the basic ones

Relational Algebra 13

Definition:

a **derived operation** in an algebraic system is an operation that is expressible in terms of standard operations of the algebra

e.g. sq( ) is derived from * via sq(x)=x*x

Derived operations on relations include

- intersection
- quotient
- join
- natural join

Relational Algebra 14

Derived relational operations …

6. Intersection of relations of same type

$R \cap S \equiv R - (R - S)$ defines tuples common to R and S

7. Quotient

$R / S \equiv$ "inverse of cartesian product"

specifies T where $T \times S = R$, when such T exists!

In general, $R / S \equiv$ set of tuples t such that <t, s> (that is, "t concatenated with s") is in R for all s in S

Relational Algebra 15

Derived relational operations …

8. Join

A **join** of R and S is defined as the subset of $R \times S$ for which there is an arithmetic relation ($<$, Ω, $=$,Δ , $>$) between the i-th component of R and the j-th component of S

Most important kind of join is the **equijoin**

$$R \underset{i=j}{*} S \equiv \sigma_{i=j}(R \times S)$$

A join is a selection from Cartesian product

Relational Algebra 16

Derived relational operations …

In practice, Cartesian product often generates relations that are too large to be computed efficiently

More practical operation to join relations is **natural join.**

Definition of natural join refers to equality of domains

$\Rightarrow$ simplest to describe w.r.t. named attributes

natural join = "equijoin without duplicate columns"

Relational Algebra 17

Derived relational operations …

9. The Natural Join

Derive the natural join $R * S$ by

- forming product $R \times S$
- selecting those tuples (r,s) where r and s have same values for all common attributes
- making a projection to remove duplicate columns that correspond to these common attributes

$R * S = \prod_{i(1),\, i(2),\, \ldots,\, i(m)} \sigma_{\Lambda(r.x=s.x)}(R \times S)$

with an appropriate choice of indices i(j) & attributes x

Summary of Relational Algebra concepts

Primitive operations:

• 1. Union	$R \cup S$
• 2. Set Difference	$R - S$
• 3. Cartesian Product	$R \times S$
• 4. Projection	$\prod_{i(1), i(2), ..., i(t)} (R)$
• 5. Selection	$\sigma_F(R)$

Derived operations: intersection, natural join, quotient

Codd's definition of **completeness:**

a query language is **complete** if it can simulate all 5 basic operations on relations

ISBL: A Relational Algebra Query Language 1

ISBL - Information System Base Language

Devised by Todd in 1976

IBM Peterlee Relational Test Vehicle (PRTV)

PL/1 environment with query language ISBL

One of the first relational query languages

… closely based on relational algebra

The six basic operations in ISBL are union, difference, intersection, natural join, projection and selection

ISBL: A Relational Algebra Query Language 2

Operators in ISBL are '+', '-', '%', ':' and '*' .

R+S	union of relations
R - S	difference operation with extended semantics
R % A, B, ... , Z	projection onto named attributes
R : F	selection of tuples subject to boolean formula F
R . S	intersection
R * S	natural join

R - S is defined *whenever R and S have some attribute names in common*: delete tuples from R that agree with S on all common attributes.

ISBL: A Relational Algebra Query Language 3

Comparison: Relational Algebra vs ISBL

$R \cup S$	R+S
$R - S$	R-S subsumes
$R \times S$	no direct counterpart
$\prod_{i(1), i(2), ..., i(t)} (R)$	R % A, B, ... , Z
$\sigma_F(R)$	R : F
contrived derived op	R * S

To prove completeness of ISBL, enough to show that can express Cartesian product using the ISBL operators - return to this issue later

ISBL: A Relational Algebra Query Language 4

ISBL as a query language

Two types of statement in ISBL

LIST <exp>	print the value of exp
R = <exp>	assign value of exp to relation R

In this context, R is a variable whose value is a relation

Notation: use R(A,B,...,Z) to refer to a relation with attributes A, B, ..., Z

ISBL: A Relational Algebra Query Language 5

Example ISBL query to specify the composition of two binary relations R(A,B) and S(C,D) where A,B,C,D are attributes defined over the same domain X (as when defining composition of functions X❑X):

Specify composition of R and S as RCS, where

RCS = (R * S) : B=C % A, D

In this case: R * S = R × S because attribute names (A, B), (C, D) are disjoint [cf. completeness of ISBL]

Illustrates archetypal form of query definition:

projection of selection of join

ISBL: A Relational Algebra Query Language 6

Assignment and call-by-value

After the *assignment*

RCS = (R * S) : B=C % A, D

the variable RCS retains its assigned value whatever happens to the values of R and S

Hence all subsequent "LIST RCS" requests obtain same value until reassignment

cf *call-by-value* parameter passing mechanisms

ISBL: A Relational Algebra Query Language 7

Delayed evaluation and call-by-name

- have a **delayed evaluation** mechanism to change the semantics of assignment cf. a "definitive notation" or a spreadsheet definition
- to delay the evaluation of the relation named R in an expression, use N!R in place of R

  RCS = (N!R * N!S) : B=C % A, D
- this means that the variable RCS is evaluated on a *call-by-name* basis: i.e. it's value is computed as required using the current values of R and S
- whenever the user invokes "LIST RCS" in this case, the value of RCS is re-computed

ISBL: A Relational Algebra Query Language 8

Uses for delayed evaluation

- definition of **views** is facilitated
- allows incremental definition of complex expressions: use sub-expressions with temporary names, supply extensional part later
- useful for optimisation: assignment means immediate computation at every step, delayed evaluation allows intelligent updating of values

ISBL: A Relational Algebra Query Language 9

Renaming

For union & intersection, attribute names must match
e.g. R(A,B) + S(A,C) is undefined etc.

To overcome this can rename attributes of R by

$(R\%A, B \rightarrow C)$

This project-and-rename creates relation R(A,C).

Can use this to make attributes of R & S disjoint, so that

$$R * S = R \times S,$$

proving that ISBL is a complete query language

Tensions between theory and practice in ISBL

- Mathematical relations abstract away certain characteristics of data that are important to the human interpreter – e.g. types, order for table inspection
- Certain activities that are an essential part of data processing, such as updating relations, forming aggregates etc are not easy to describe formally
- Classical algebra uses homogeneous data types, doesn’t deal elegantly with exceptions 3/0 = ? etc

ISBL: A Relational Algebra Query Language 10

Limitations of ISBL

ISBL is complete, but lacks features of QUEL, SQL etc
e.g. no aggregate operators
no insertion, deletion and modification
Primarily a declarative query language

Address these issues in the PRTV environment - user can also access relations via the general-purpose programming language PL/1

ISBL: A Relational Algebra Query Language 11

Illustrative examples of ISBL use

Refer to the Happy Valley Food Company [Ullman 82]

Relations in this DB are:

MEMBERS(NAME, ADDRESS, BALANCE)
ORDERS(ORDER_NO, NAME, ITEM, QUANTITY)
SUPPLIERS(SNAME, SADDRESS, ITEM, PRICE)

ISBL: A Relational Algebra Query Language 12

Illustrative examples of ISBL use

MEMBERS(NAME, ADDRESS, BALANCE)
ORDERS(ORDER_NO, NAME, ITEM, QUANTITY)
SUPPLIERS(SNAME, SADDRESS, ITEM, PRICE)

1. Print the names of members in the red:

```
LIST MEMBERS : BALANCE < 0 % NAME
```

i.e. select members with negative balance and project out their names

ISBL: A Relational Algebra Query Language 13

Illustrative examples of ISBL use

MEMBERS(NAME, ADDRESS, BALANCE)
ORDERS(ORDER_NO, NAME, ITEM, QUANTITY)
SUPPLIERS(SNAME, SADDRESS, ITEM, PRICE)

2. Print the supplier names, items & prices for suppliers who supply at least one item ordered by Brooks

```
OS = ORDERS * SUPPLIERS
LIST OS: NAME="Brooks" % SNAME, ITEM, PRICE
```

... a simple example of project-select-join

ISBL: A Relational Algebra Query Language 14

Illustrative examples of ISBL use

MEMBERS(NAME, ADDRESS, BALANCE)
ORDERS(ORDER_NO, NAME, ITEM, QUANTITY)
SUPPLIERS(SNAME, SADDRESS, ITEM, PRICE)

2. (commentary on answer) Need two of the relations:
   SUPPLIERS required for supplier details
   ORDERS to know what Brooks has ordered

The join OS holds tuples where item field contains item *"ordered with associated order info"* and *"supplied by supplier with assoc supplier info"*

… tuples featuring Brooks' name correspond to an item ordered by Brooks with its associated supplier details

ISBL: A Relational Algebra Query Language 15

3. Print suppliers who supply *every* item ordered by Brooks

"Every item" is universal quantification

Strategy: translate $(\forall x)(p(x))$ to $\neg(\exists x)(\neg p(x))$
find suppliers who don't supply at least one of the items that is ordered by Brooks, and take the complement of this set of suppliers

Notation: $\forall$ is "for all", $\exists$ is "there exists", $\neg$ is "not"

ISBL: A Relational Algebra Query Language 16

3. ... suppliers supplying *every* item ordered by Brooks

S = SUPPLIERS % SNAME
I = SUPPLIERS % ITEM
NS = (S * I) - (SUPPLIERS % SNAME, ITEM)

- S records all supplier names, and I all items supplied
- NS is the "does not supply" relation: all supplier-item pairs with pairs such that s supplies i eliminated

Now specify items ordered by Brooks ...
B = ORDERS : NAME="Brooks" % ITEM

ISBL: A Relational Algebra Query Language 17

3. … suppliers supplying *every* item ordered by Brooks

NS = "doesn't supply" relation
B = "items ordered by Brooks"

... find suppliers who don't supply at least one item in B
NSB = NS.(S * B)
.... set of (supplier, item) pairs such s doesn't supply i and Brooks ordered i.

Answer is the complement of this set:
S - NSB % SNAME

To follow …

Relational Theory: Algebra and Calculus
SQL review